Module 3
Repetition Statements

Tamer Abdou, PhD

IND 830 - Python Programming for Data Science

Data Science Laboratory

Ryerson University

Ryerson University
The Chang School of Continuing Education

Outline

1. *Definite Iterations*
2. *Indefinite Iterations*
3. *Nested Iterations*

Definite Iterations

1 Definite Iterations
- Count-Controlled Loops
- Repeating a Sequence of Actions
- Augmented Assignment
- Traversing a Sequence of Actions

2 Indefinite Iterations

3 Nested Iterations

Count-Controlled Loops

- The count-controlled loops are loops that iterate through a specific range of numbers
- They can be used to:
  - Repeat a sequence of actions a fixed number of times
  - Traverse a sequence of characters in a string or a complex data structure
  - Count up/down from an initial value to a final limit
- These loops require:
  - A loop counter
  - Initial and Final values
  - A step value

Repeating Actions Using Simple `range()`

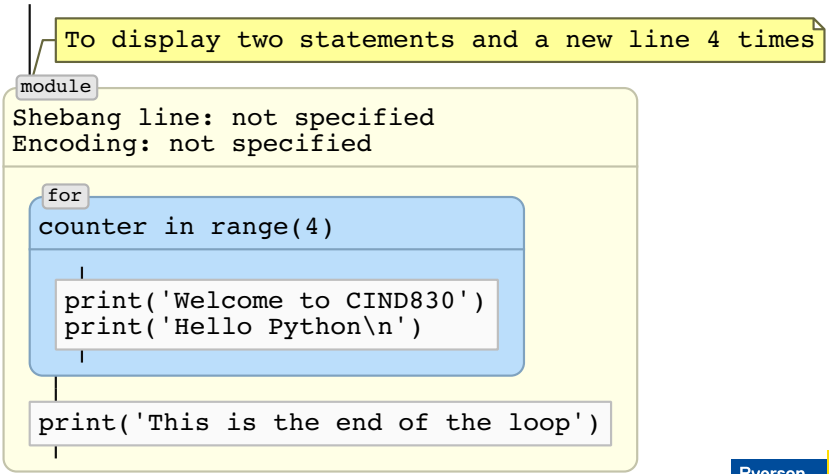
```
for <variable> in range(<an integer expression>):
    <statement-1>
     <...>
    <statement-n>
```

- The **integer expression** in the **loop header** indicates the number of iterations that the loop performs
- The **colon (:)** ends the loop header
- The **loop body** contains the set of statements which will be repeated in sequence on each iteration through the loop
- The loop body must be **indented** and aligned in the same column

Example 3.1 (Script)

```
1 #To display two statements and a new line 4 times
2 for counter in range(4):
3     print('Welcome to CIND830')
4     print('Hello Python\n')
5 print('This is the end of the loop')
```

Example 3.1 (Control Flow)



Example 3.1 (Program Execution)

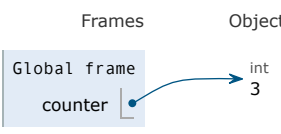
Write code in Python 3.6

(drag lower right corner to resize code editor)

```
1  #To display two statements
2  #and a new line 4 times
3  for counter in range(4):
4      print('Welcome to CIND830')
5      print('Hello Python\n')
6  print('This is the end of the loop')
7
```

Print output (drag lower right corner to resize)

```
Welcome to CIND830
Hello Python

Welcome to CIND830
Hello Python

Welcome to CIND830
Hello Python

Welcome to CIND830
Hello Python

This is the end of the loop
```



Repeating Actions Using Extended `range()`

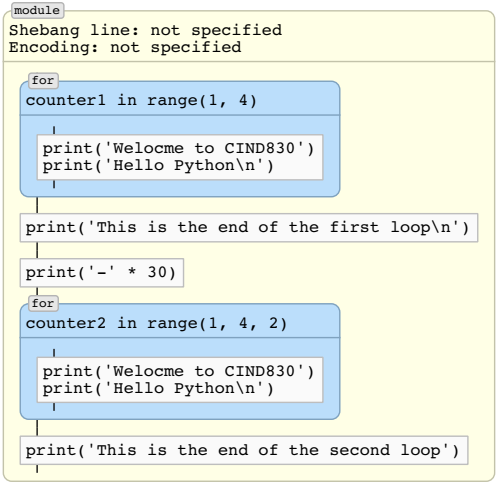
```
for <var> in range(<lowerBound>,<upperBound+1>, stepValue):
    <loop body>
```

- The count starts from 0 to the value of the integer expression minus 1, or from the lower bound to the upper bound
- These values are used in computations in the loop body
- The step value is an optional argument, which specifies the interval between the numbers used in the range
- If the lower bound is higher than the upper bound, then the step value should be a negative integer and the loop will be counting down

Example 3.2 (Script)

```
1  for counter1 in range(1, 4):
2      print('Welocme to CIND830')
3      print('Hello Python\n')
4  print('This is the end of the first loop\n')
5
6  print('-' * 30)
7
8  for counter2 in range(1, 4, 2):
9      print('Welocme to CIND830')
10     print('Hello Python\n')
11 print('This is the end of the second loop')
```

Example 3.2 (Control Flow)



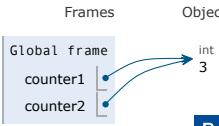
Example 3.2 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  for counter1 in range(1, 4):
2      print('Welocme to CIND830')
3      print('Hello Python\n')
4  print('This is the end of the first loop\n')
5
6  print('-' * 30)
7
8  for counter2 in range(1, 4, 2):
9      print('Welocme to CIND830')
10     print('Hello Python\n')
11 print('This is the end of the second loop')
```

Print output (drag lower right corner to resize)

```
Welocme to CIND830
Hello Python

Welocme to CIND830
Hello Python

Welocme to CIND830
Hello Python

This is the end of the first loop

------------------------------
Welocme to CIND830
Hello Python

Welocme to CIND830
Hello Python

This is the end of the second loop
```



Augmented Assignment

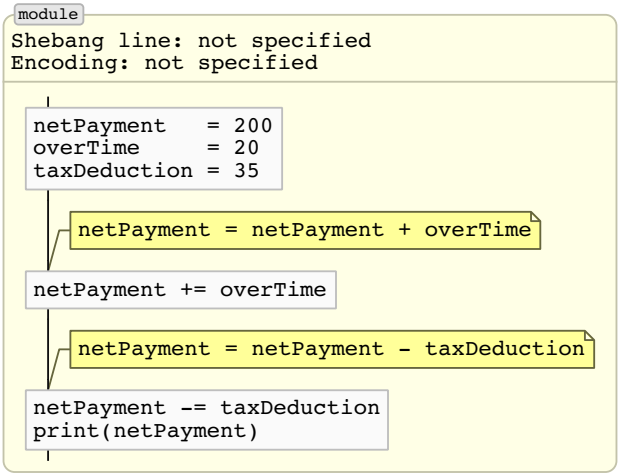
```
<variable> <operator>= <expression>
is equivalent to
<variable> = <variable> <operator> <expression>
```

- The assignment symbol (=) can be combined with the arithmetic operators to provide augmented assignment operations
- $x+=1$ is equal to $x=x+1$
- Note that there is no space between the operator and the assignment symbol

Example 3.3 (Script)

```
1  netPayment   = 200
2  overTime     = 20
3  taxDeduction = 35
4
5  # netPayment = netPayment + overTime
6  netPayment += overTime
7  # netPayment = netPayment - taxDeduction
8  netPayment -= taxDeduction
9  print(netPayment)
```

Example 3.3 (Control Flow)



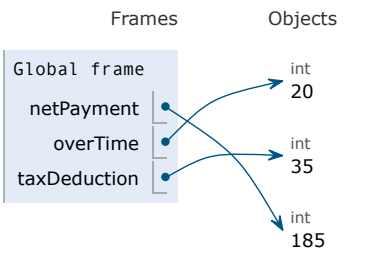
Example 3.3 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  netPayment = 200
2  overTime = 20
3  taxDeduction = 35
4  #---------------------------
5  # netPayment = netPayment + overTime
6  netPayment += overTime
7  # netPayment = netPayment - taxDeduction
8  netPayment -= taxDeduction
9  print(netPayment)
```

Print output (drag lower right corner to resize)

```
185
```



Traversing the Contents of a Data Sequence

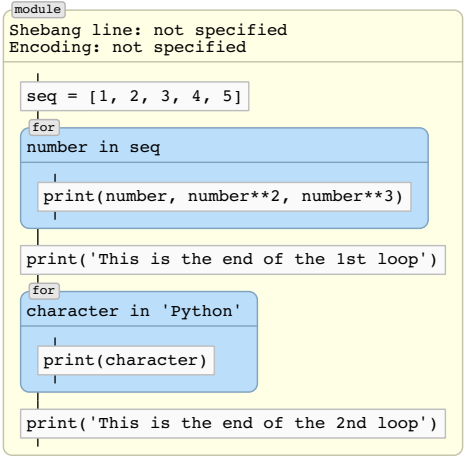
```
for <variable> in <sequence>:
    <do something with the variable use the print statement>
```

- The variable here can be a number or a character
- In each iteration, the next value in the sequence is assigned to the variable

Example 3.4 (Script)

```
1  seq = [1, 2, 3, 4, 5]
2  for number in seq:
3      print(number, number**2, number**3)
4  print('This is the end of the 1st loop')
5
6  for character in 'Python':
7      print(character)
8  print('This is the end of the 2nd loop')
```

Example 3.4 (Control Flow)



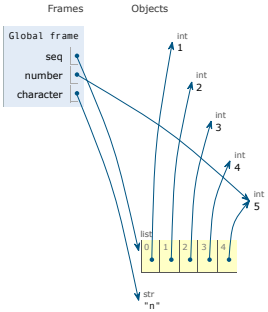
Example 3.4 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1 seq = [1, 2, 3, 4, 5]
2 for number in seq:
3     print(number, number**2, number**3)
4 print('This is the end of the 1st loop')
5
6 for character in 'Python':
7     print(character)
8 print('This is the end of the 2nd loop')
```

Print output (drag lower right corner to resize)

```
(1, 1, 1)
(2, 4, 8)
(3, 9, 27)
(4, 16, 64)
(5, 25, 125)
This is the end of the 1st loop
P
y
t
h
o
n
This is the end of the 2nd loop
```



Indefinite Iterations

1 Definite Iterations

2 Indefinite Iterations
Condition-Controlled Loops
The while Loop

3 Nested Iterations

Condition-Controlled Loops

- The condition-controlled loops are loops that iterate without knowing in advance the number of times the loop will be executed
- They can be used to:
  - Process data from a file without knowing in advance the end of that file
  - Traverse a sequence of actions while a certain condition remains true
  - Count up/down from an initial value until a given condition changes
- These loops require:
  - A loop counter
  - A logical condition

The Conditional `while` Loop

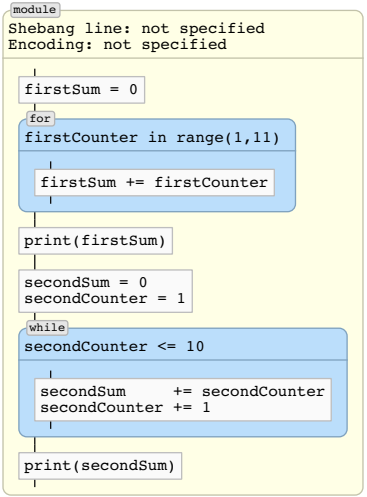
```
while <condition>:
      <sequence of statements>
```

- The condition is tested within the loop to determine whether the loop should continue
- As long as the continuation condition is **True**, the sequence of statements within the loop will be executed
- Once the condition is **False**, the loop ends
- Inside the loop, a variable needs to be adjusted for the condition to be **False**, otherwise the loop will continue forever
- Note that, the sequence of statements can be executed zero or more times, as it depends on the condition which is tested at the beginning of the loop

Example 3.5 (Script)

```
1  firstSum = 0
2  for firstCounter in range(1,11):
3      firstSum += firstCounter
4  print(firstSum)
5
6  secondSum = 0
7  secondCounter = 1
8  while secondCounter <= 10:
9      secondSum     += secondCounter
10     secondCounter += 1
11 print(secondSum)
```

Example 3.5 (Control Flow)



Example 3.5 (Program Execution)

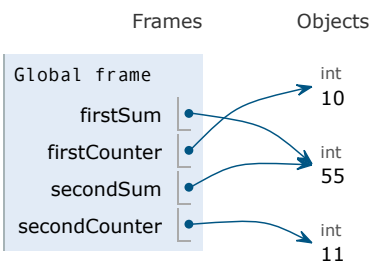
Write code in Python 3.6

(drag lower right corner to resize code editor)

```
1  firstSum = 0
2  for firstCounter in range(1,11):
3      firstSum += firstCounter
4  print(firstSum)
5
6  secondSum = 0
7  secondCounter = 1
8  while secondCounter <= 10:
9      secondSum     += secondCounter
10     secondCounter += 1
11 print(secondSum)
```

Print output (drag lower right corner to resize)

```
55
55
```



Nested Iterations

1 Definite Iterations

2 Indefinite Iterations

3 Nested Iterations
Loops inside Loops
Application

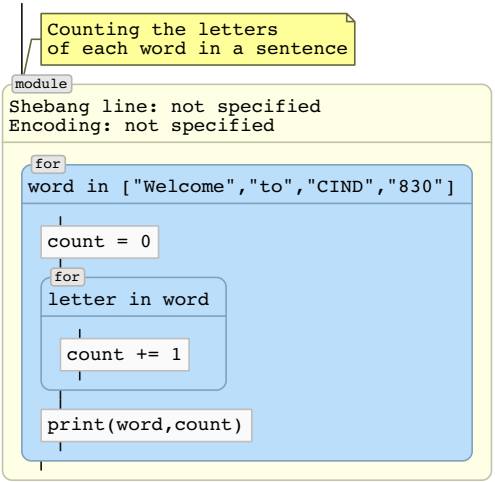
Loops inside Loops

- The nested loops are loops that iterate within other loops
- They can be used to:
  - Repeat a sequence of actions inside another loop
  - Traverse a set of strings and for each string count how many times a certain character is found
- These loops require:
  - An outer loop
  - An inner loop

Example 3.6 (Script)

```
1  #Counting the letters
2  #of each word in a sentence
3  for word in ["Welcome","to","CIND","830"]:
4      count = 0
5      for letter in word:
6          count += 1
7      print(word,count)
```

Example 3.6 (Control Flow)



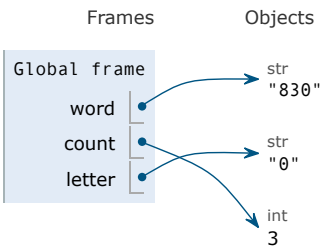
Example 3.6 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  #Counting the letters
2  #of each word in a sentence
3  for word in ["Welcome","to","CIND","830"]:
4      count = 0
5      for letter in word:
6          count += 1
7      print(word,count)
```

Print output (drag lower right corner to resize)

```
('Welcome', 7)
('to', 2)
('CIND', 4)
('830', 3)
```



References

Lambert, K. A. (2010). Control Statements. In *Fundamentals of Python: From first programs through data structures*(pp.76-90, 102-115). Boston, MA: Cengage Learning.